colloquium exactarum

# LEPATTERN - UMA FERRAMENTA PARA ENSINO DE PADRÕES DE PROJETO: IMPLEMENTAÇÃO DOS PADRÕES STRATEGY E FACADE.

## LEPATTERN - A LEARNING TOOL TO DESIGN PATTERNS: IMPLEMENTATION OF STRATEGY AND FACADE.

Caroline Teixeira da Cruz[1], Mario Augusto Pazoti[2], Francisco Virginio Maracci[3]

Faculdade de Informática de Presidente Prudente - Universidade do Oeste Paulista;
caroline@hotmail.com[1]; mpazoti@gmail.com[2]; francisco@unoeste.br[3]

**RESUMO –** Padrões de projeto são aplicados com o intuito de reutilizar soluções padronizadas de software em problemas corriqueiros. No entanto, há ainda um obstáculo: a dificuldade no aprendizado sobre padrão de projeto e a falta de conhecimento dos projetistas envolvidos. O propósito deste trabalho é fornecer um módulo de extensão (*plugin*) que realize a implementação da estrutura de alguns padrões de projeto com o intuito de auxiliar no aprendizado e na sua aplicação durante o desenvolvimento de software. O resultado deste trabalho apresenta comentários associados explicando a funcionalidade de cada etapa de criação das classes referentes ao projeto. Conclui-se que utilizando de forma correta, poderá haver redução de custo e de tempo, em todo o ciclo de vida do software e ainda o uso do *plugin* auxilia de forma significativa na aprendizagem e implementação dos padrões.
**Palavras-chave:** Padrão de Projeto; Engenharia de Software; Reúso.

**ABSTRACT -** Design patterns are applied to reuse standardized software solutions to solve common problems. However, there is still an obstacle: the difficulty in learning about design pattern and the lack of knowledge of the involved designers. The purpose of this paper is to provide an extension module (plugin) to conduct the implementation of the structure of some design patterns with the help of order in learning and its implementation in the software development. The result of this work has associated comments explaining the functionality of each of the classes related to the creation phase design. It follows that by using properly, there may be a reduction of cost and time, in the entire software life cycle and also the use of plug assist significantly in the implementation of learning and pattern.
**Keywords:** Design Pattern; Software Engineering; Reuse.

Recebido em: 15/04/2015
Revisado em: 07/10/2015
Aprovado em: 03/11/2015

## 1 INTRODUÇÃO

Com a tecnologia cada vez mais presente no cotidiano da sociedade, tornou-se comum o uso de sistemas computadorizados para realizar tarefas que antes eram feitas manualmente. Porém, durante o processo de desenvolvimento é comum os projetos apresentarem problemas recorrentes. Desse modo, sempre haverá o trabalho de resolvê-los mais de uma vez, entretanto, se a solução já for conhecida, reutilizá-la aceleraria bastante o processo de desenvolvimento.

A partir disso, foram criados padrões de projetos de software, que consistem, basicamente, em uma forma de reutilizar soluções padronizadas de software para problemas corriqueiros. Porém, apesar de existirem vários padrões de projetos na literatura, como os de Gamma et al. (1995), ainda existe a dificuldade em seu aprendizado e a falta de conhecimento dos projetistas envolvidos.

Segundo Della e Clark (2000), compreender padrões é difícil e eles concluíram que essa dificuldade vem da falta de conhecimento e experiência com tecnologias orientada a objetos. Partindo destas informações, neste trabalho é apresentado um módulo de extensão (*plugin)* que foi desenvolvido para facilitar o aprendizado quanto a aplicação de padrões de projetos, uma vez que a partir do

padrão selecionado será construída toda a estrutura de forma fácil e direcionada.

Como existem muitos padrões de projeto, é necessário que haja uma forma de organizá-los. A classificação ajuda a aprender os padrões mais rapidamente, direcionando esforços na descoberta de novos padrões. O catálogo de padrões apresentado por Gamma et al. (1995) possui vinte e três padrões distintos.

A classificação dos padrões projetos é feita por dois critérios, sendo que o primeiro é a finalidade ou propósito. O propósito é dividido em: de criação, estrutural, comportamental. O segundo critério é chamado de escopo.

Na Tabela 1 é apresentada a classificação dos padrões de projeto de acordo com esses dois critérios.

**Tabela 1.** Catalogo de padrões de projeto de Gamma et al. (1995).

| | | **Propósito** | | |
|---|---|---|---|---|
| | | De Criação | Estrutural | Comportamental |
| **Escopo** | **Classe** | Factory Method | Adapter | Interpreter<br>Template Method |
| | **Objeto** | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Adapter<br>Bridge<br>Composite<br>Decorator<br>Façade<br>Flyweygth<br>Proxy | Chain of Resposability<br>Command<br>Interator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

O escopo reflete à classe ou a os objetos em que o padrão poderá ser aplicado. Os padrões de classe lidam com o relacionamento por meio de herança com a classe e suas subclasses em tempo de compilação, deste modo são estáticos. Já padrões de objeto lidam com relacionamento entre objetos podendo ser alterados em tempo de execução, e por isso são dinâmicos (GUERRA, 2013).

De maneira geral Gamma et al. (1995) descreve que os padrões de criação tem a finalidade de especificar técnicas para instanciar e criar objetos, abstraindo o processo de instanciação das classes, ajudando o sistema ser independe dos objetos concretos criados.

Os padrões estruturais, segundo Gamma et al. (1995), descrevem formas de tratamento entre classes e objetos em sistemas, possibilitando uma formação de estruturas maiores e complexas. Padrões estruturais de classe realizam herança de uma interface a fim de criar novas interfaces, proporcionando assim um trabalho de forma independente. Padrões estruturais de objeto descrevem maneiras de compor objetos para realizar uma nova funcionalidade, possibilitando assim uma flexibilidade de manusear objetos em tempo de execução.

Padrões comportamentais se preocupam com algoritmos e atribuição de responsabilidades entre os objetos, prescrevendo a comunicação entre objetos e classes. Esses padrões de classe utilizam a herança para distribuir o comportamento entre as classes, enquanto os padrões de objetos utilizam composições de objetos e, por isso, se complementam e se reforçam mutuamente (GAMMA et al., 1995).

A relevância deste trabalho se dá pela disponibilização de tecnologia da informação aos profissionais por meio da implementação das soluções propostas pelos padrões de projeto. Ao implementar a estrutura de dois padrões de projetos de software especificado por Gamma et al. (1995): strategy e facade.

O objetivo é auxiliar o aprendizado de padrões em situações triviais, além de apresentar as vantagens na utilização de padrões de projeto para o desenvolvimento de sistemas.

Para tanto, a ferramenta proposta apresenta a teoria a respeito dos padrões codificados, bem como exemplos de aplicação dos mesmos. Permite, ainda, que o usuário durante o aprendizado configure informações a respeito dos padrões e classes pertencentes para que um código-fonte automático e comentado seja gerado. Dessa forma, o desenvolvedor terá acesso a toda estrutura implementada realizando seu uso de acordo com o contexto a que for exposto.

As demais Seções deste artigo estão organizadas da seguinte maneira: na Seção 2 é apresentada a metodologia utilizada no desenvolvimento do trabalho; na Seção 3 é apresentada o desenvolvimento do trabalho; na Seção 4 apresentada os experimentos realizados; na Seção 5 apresentada a análise dos resultados.

## 2 METODOLOGIA

As duas primeiras fases do projeto consistiram em realizar um estudo dos padrões de projeto e a codificação de um *plugin* na linguagem de programação Java. Após o término do desenvolvimento dessas etapas, foi realizada uma análise qualitativa, a partir do desenvolvimento de sistemas utilizando os padrões de projetos disponibilizados no *plugin*.

O desenvolvimento foi realizado em um *plugin*, pois além de ter um tamanho reduzido ele está mais próximo de um ambiente de programação, o que torna seu uso mais fácil e prático. O uso de plugin permitiu que o código-fonte gerado automaticamente pela ferramenta seja adicionado diretamente no diretório do projeto da aplicação em desenvolvimento.

Os padrões de projeto implementados no *plugin* foram Strategy e Facade. O padrão Strategy é utilizado para encapsular algoritmos semelhantes a fim de definir novas operações sem alterar as classes dos elementos sobre os quais opera.

Gamma et al. (1995) afirma que a grande vantagem do uso deste padrão é que além de permitir reutilização de código, as funcionalidades que compõe o projeto são facilmente estendidas e as manutenções no projeto são menos custosas.

Pode ser visualizado o diagrama de classes do modelo de implementação do padrão *strategy* na Figura 1.

Analisando a figura 1, note que a classe Context contém uma referência ao objeto Strategy e pode definir uma interface que permite a estratégia de acesso aos seus dados (GUERRA, 2013).

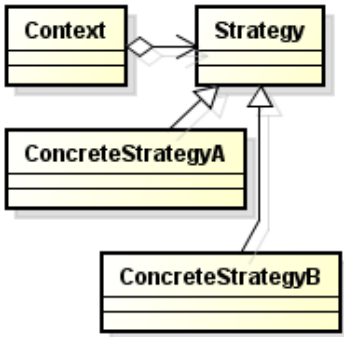


**Figura 1.** Estrutura de Classes do Padrão Strategy (GAMMA et al., 1995).

Na classe Strategy é definida uma interface comum a todos os algoritmos. A classe Context utiliza esta interface para instanciar o algoritmo definido por ConcreteStrategy. Já na classe ConcreteStrategy será implementado um algoritmo para cada estratégia.

O padrão Facade é aplicado em projetos em que as classes principais dependem do acesso das diversas funcionalidades do sistema sendo necessário o conhecimento detalhado de todo o subsistema para usá-lo (FREEMAN; FREEMAN, 2009).

Assim, o padrão Facade tem a função de fornecer uma interface simples para comportamento de um conjunto de objetos e que abrange ações repetitivas comuns ao sistema. Desse modo, o padrão é capaz de reduzir a complexidade do sistema, uma vez que o sistema irá integrar somente com a interface facade que contém a sequencia de objetos e possibilita o acesso ao comportamento do sistema. Pode ser visualizado o diagrama de classe do modelo de implementação do padrão Facade na Figura 2 (FREEMAN; FREEMAN, 2009).

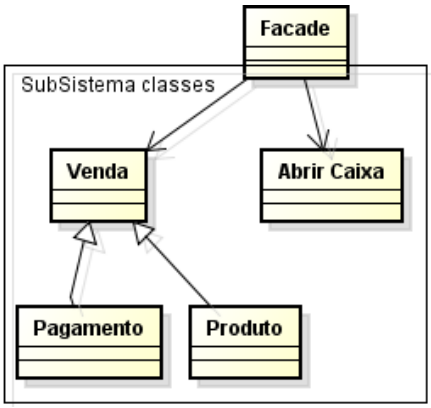


**Figura 2.** Estrutura de Classes do Padrão Facade (GAMMA et al., 1995)

A classe Facade define quais classes do subsistema são responsáveis por um determinado pedido, delega solicitação de cliente para os subsistemas apropriados (GUERRA, 2013).

As classes que pertence ao Subsistema implementam a funcionalidade especifica do subsistema ao realizar as operação designada pelo objeto Facade (FREEMAN; FREEMAN, 2009).

## 3 DESENVOLVIMENTO

A estrutura criada para realizar o processo de criação no *plugin* é descrita na Figura 3.


**Figura 3.** Sequência de execução para a criação da estrutura no plugin

No item inserir dados apresentado na Figura 3, irá realizar a seguinte operação, o usuário da ferramenta irá digitar os dados necessários descrito na ferramenta, com os dados que o mesmo deseja. Após a operação de inserir os dados, será gerado uma estrutura que tenha a garantia mínima que caracterize o padrão conforme a base de dados inserido pelo usuário. Finalizando com a criação automática do código – fonte, onde no mesmo contém comentários descrevendo a função de cada operação contida.

O projeto proporciona um resultado satisfatório quanto à geração de código do padrão escolhido. Além disso, como a ferramenta é dirigida ao ensino de padrões de projeto, seu *layout* foi elaborado com o objetivo de facilitar a aplicação, contendo comentários explicativos de como utilizar além da explicação sobre padrões no código gerado. Isso foi possível em virtude da aplicação do padrão ser classificada em etapas, as quais organizam o processo ao mesmo tempo em que apresentam orientações e conceitos relacionados.

A ferramenta foi desenvolvida para ser utilizada como um *plugin* do ambiente de programação NetBeans, em que será adicionado pelo usuário para utilização.

A ferramenta está estruturada de forma simples e intuitiva ao usuário. A partir da seleção do padrão, será exibida uma breve descrição do seu funcionamento, todas as telas contém uma explicação de como utilizar. Com resultados parciais obtidos até o momento, observam-se vantagens no uso dessa ferramenta, por proporcionar agilidade na aplicação do padrão e geração do código referente às classes necessárias. No entanto para a utilização do mesmo deve-se saber o conceito de orientação a objeto.

## 4 EXPERIMENTOS

Para a realização de testes com a ferramenta desenvolvida foram usadas dois exemplos. Será apresentada a interface para a implementação de cada padrão e o código resultante, gerado pela ferramenta.

As telas de interface que permite o fluxo de aprendizagem e configuração para o padrão Strategy são mostradas nas Figuras 4, 5, 6, 7 e 8. Na Figura 4 é apresentada a interface inicial da ferramenta para o padrão Strategy com a descrição e um exemplo problema com o respectivo diagrama de classe. A interface apresentada na Figura 5 demonstra os campos de formatação para a criação da classe Context.

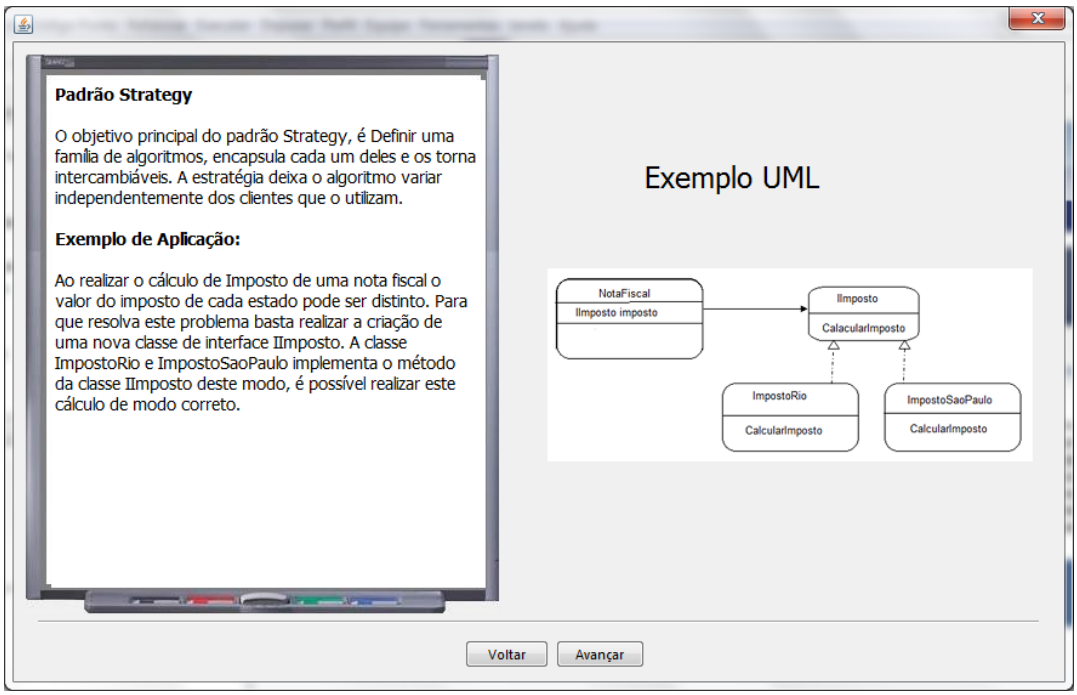


**Figura 4.** Tela do plugin para o padrão Strategy: definição do padrão.

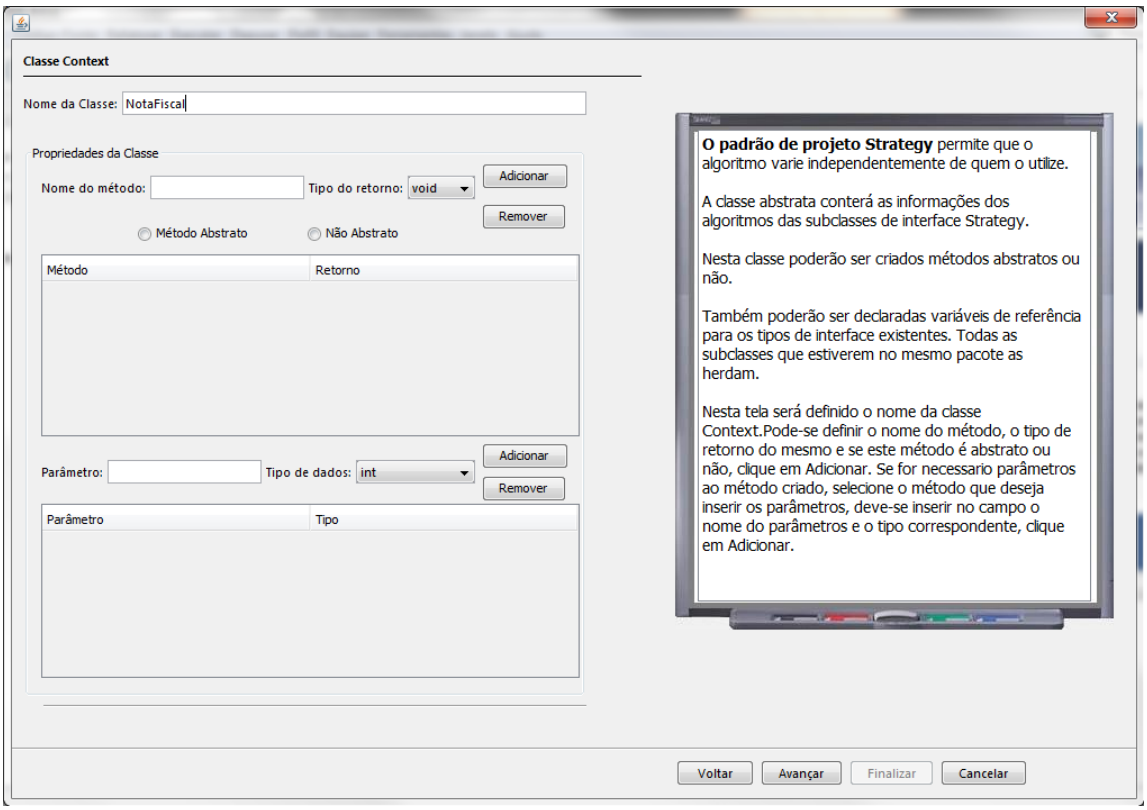


**Figura 5.** Tela do plugin para o padrão Strategy: definições da classe context.

Os dados desejados para as classes que são Encapsuladas são configurados utilizando a interface demonstrada na Figura 6. A Figura 7 apresenta a interface onde serão inseridos os dados desejados para a classe Strategy.

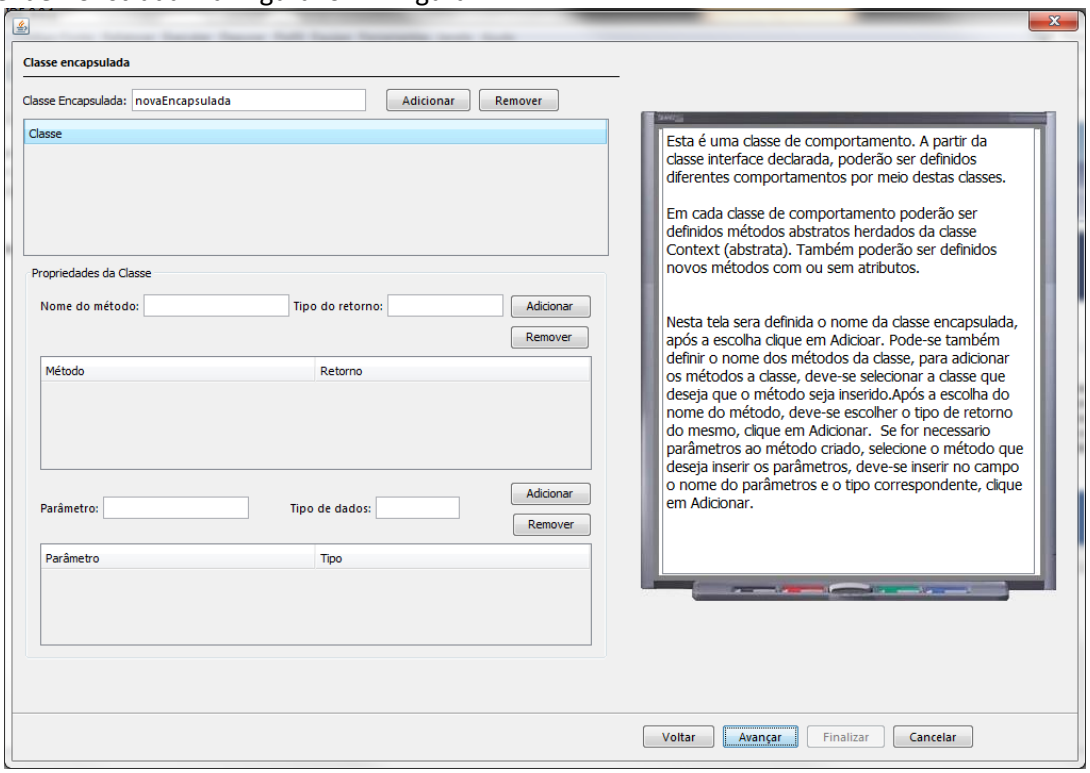


**Figura 6.** Tela do plugin para o padrão Strategy: definições da classe encapsulada.

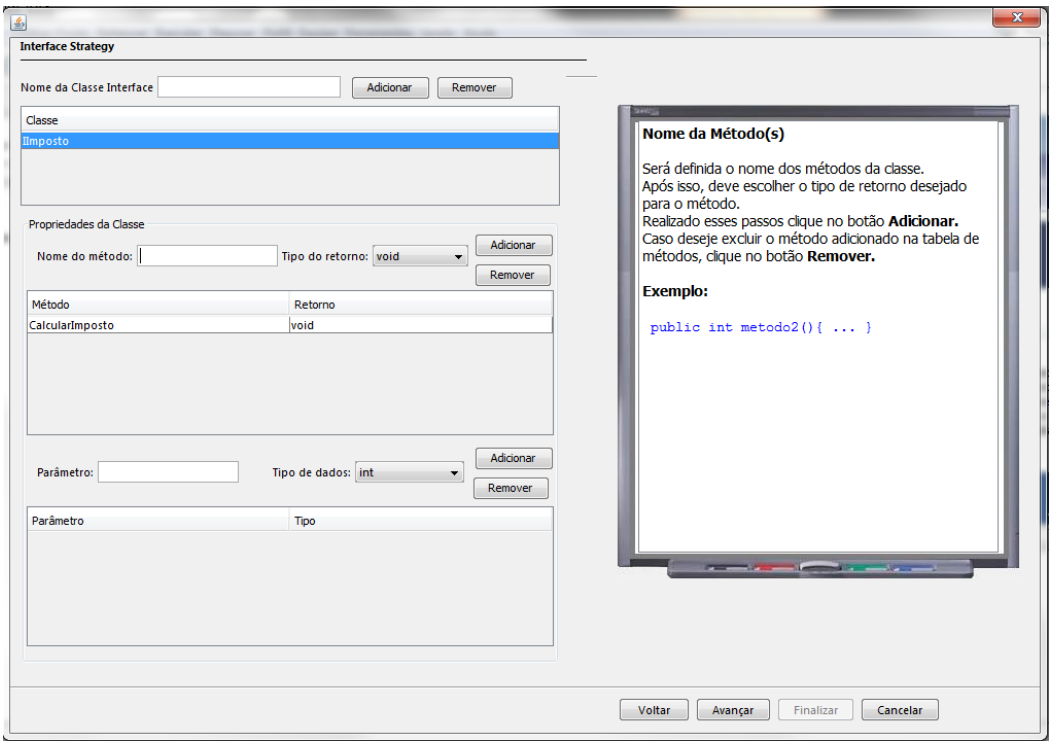


**Figura 7.** Tela do plugin para o padrão Strategy: definições da interface strategy.

Na Figura 8, serão inseridos os dados onde serão criadas as classes que irão herdar da classe Strategy.

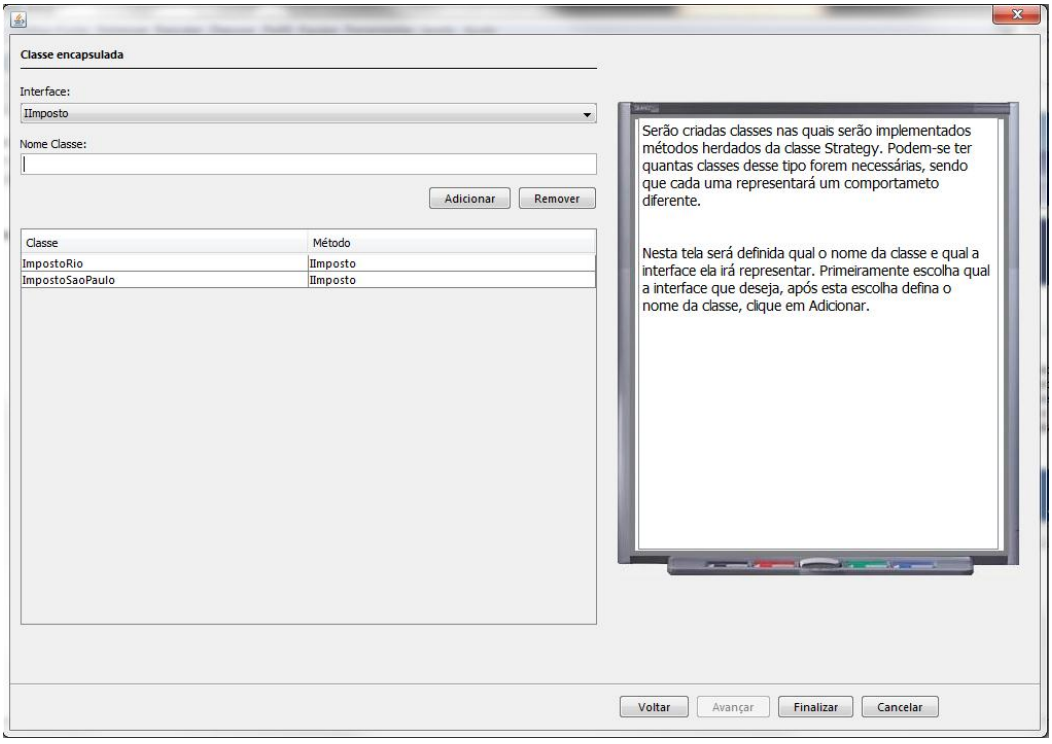


**Figura 8.** Tela do plugin para o padrão Strategy: definições da classe encapsulada.

A partir do exemplo contido na Figura 4: Ao realizar o cálculo de Imposto de uma nota fiscal o valor do imposto de cada estado pode ser distinto. Para que resolva este problema basta realizar a criação de uma nova classe de interface IImposto. A classe ImpostoRio e ImpostoSaoPaulo implementa o método da classe IImposto deste modo, é possível realizar este cálculo de modo correto. O *plugin* irá gerar os códigos automaticamente como apresentados pelas

Figuras 9, 10, 11 e 12. A figura 9 mostra o código da classe Nota Fiscal, sendo que esta é uma classe abstrata. A Figura 10, contém o código da classe do tipo interface denominada de IImposto. A classe concreta ImpostoRio e ImpostoSaoPaulo que herdam o método da interface são apresentadas nas Figuras 11 e 12 respectivamente.

```
/*Classe Context define operações especificas do Strategy.
*Contem referencia ao objeto Strategy.*/
public abstract class NotaFiscal
{
 IImposto  iimposto;
 NotaFiscal(){}
 public void setIImposto( IImposto iimposto)
 {
  this.iimposto = iimposto;
 }
 public IImposto getIImposto()
 {
  return iimposto;
 }
}
```

**Figura 9.** Código-fonte gerado para a Classe Context denominada Nota Fiscal.

```
/*Nesta classe Interface apresenta um método comum a todos os algoritmos.
Este método terá um comportamento distinto a cada classe que ele for herdado.*/
public interface IImposto
{
        public void CalcularImposto();
}
```

**Figura 10.** Código-fonte gerado para a Classe Interface denominada de IImposto.

```
/*E implementado nesta classe o método herdado da classe interface.
Cada método terá um comportamento distinto na classe que o herda. */
public class ImpostoRio implements IImposto
{
  @Override
  public void CalcularImposto()
  {
    throw new UnsupportedOperationException("Not supported yet.");
  }
}
```

**Figura 11.** Código-fonte gerado para a Classe que implementa a Interface denominada de ImpostoRio.

```
/*E implementado nesta classe o método herdado da classe interface.
Cada método terá um comportamento distinto na classe que o herda. */
public class ImpostoSaoPaulo implements IImposto
{
  @Override
  public void CalcularImposto()
  {
    throw new UnsupportedOperationException("Not supported yet.");
  }
}
```

**Figura 12.** Código-fonte gerado para a Classe que implementa a Interface denominada de ImpostoSaoPaulo.

As telas de interface que permite o fluxo de aprendizagem e configuração para o padrão Facade são demonstradas nas Figuras 13, 14, 15, 16 e 17. Na Figura 13, contém a definição do padrão e um exemplo problema com o respectivo diagrama de classe.

A Figura 14 apresenta as três formas de realizar a criação do padrão.

A próxima tela, Figura 15, permite que sejam criadas as classes do subsistema. A Figura 16, apresenta a tela que permite a criação da classe Facade. A próxima tela, Figura 17, permite que seja criado o método simplificador.

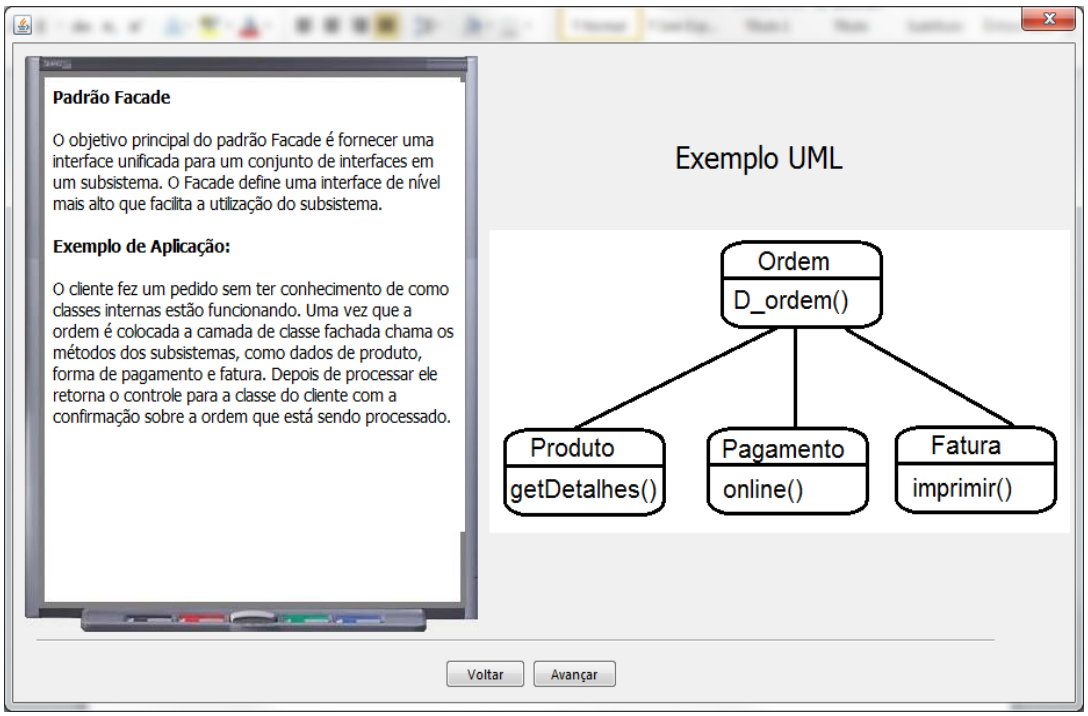


**Figura 13.** Tela do plugin para o padrão Facade: definição do padrão.

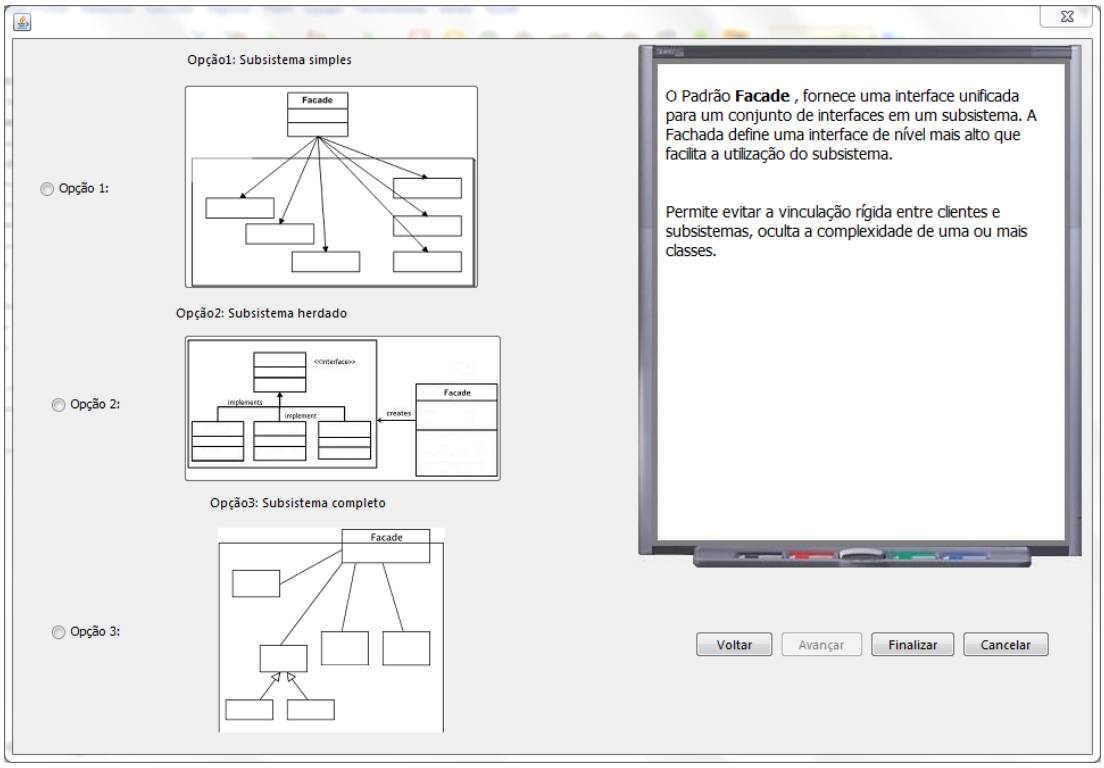


**Figura 14.** Tela do plugin para o padrão Facade: formas de criação.

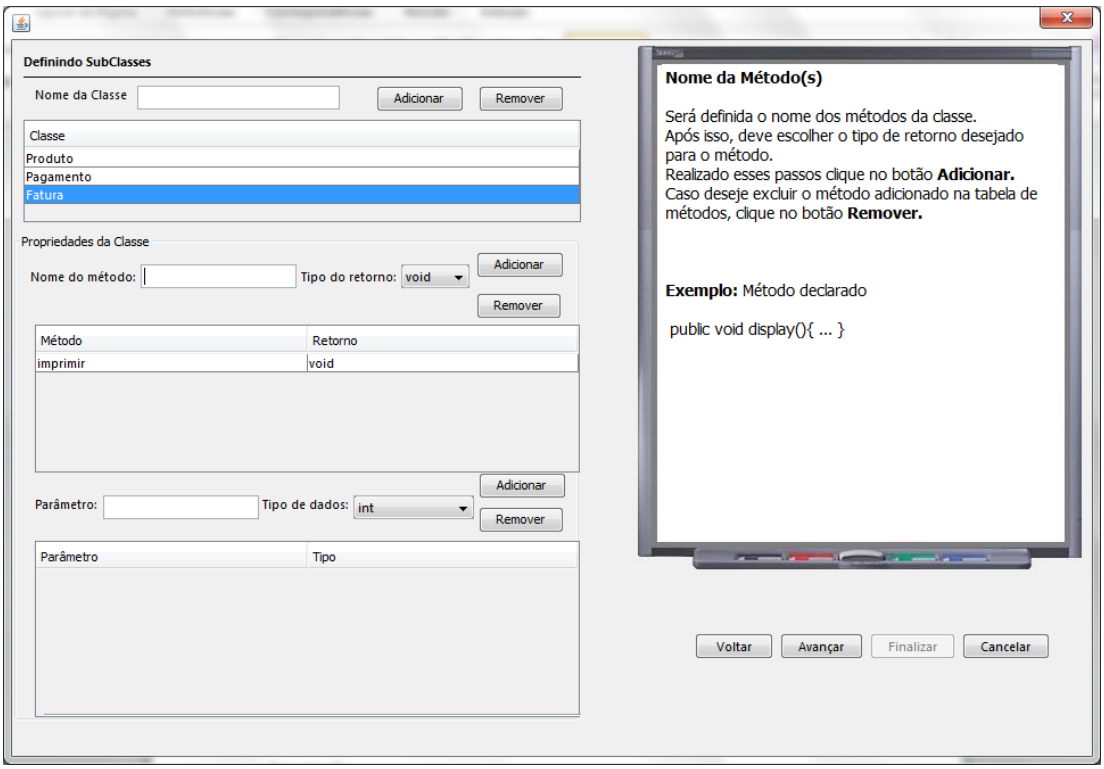


**Figura 15.** Tela do plugin para o padrão Facade: definições das classes do subsistema.

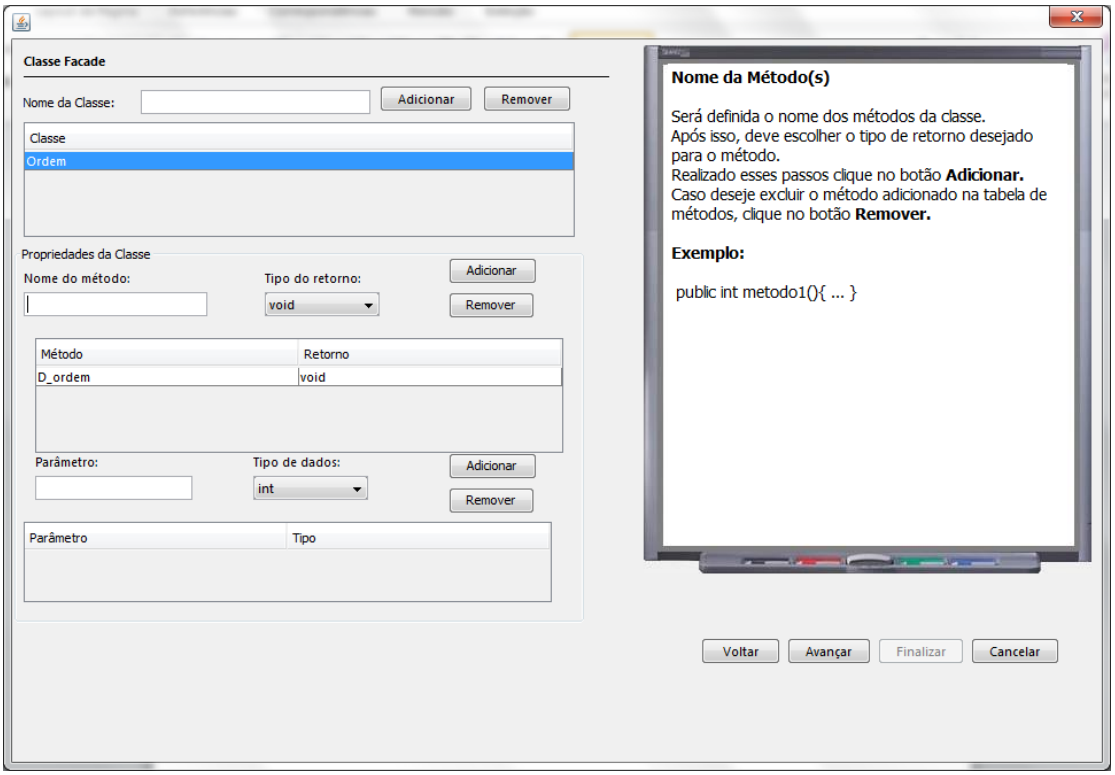


**Figura 16.** Tela do plugin para o padrão Facade: definições da classe facade.

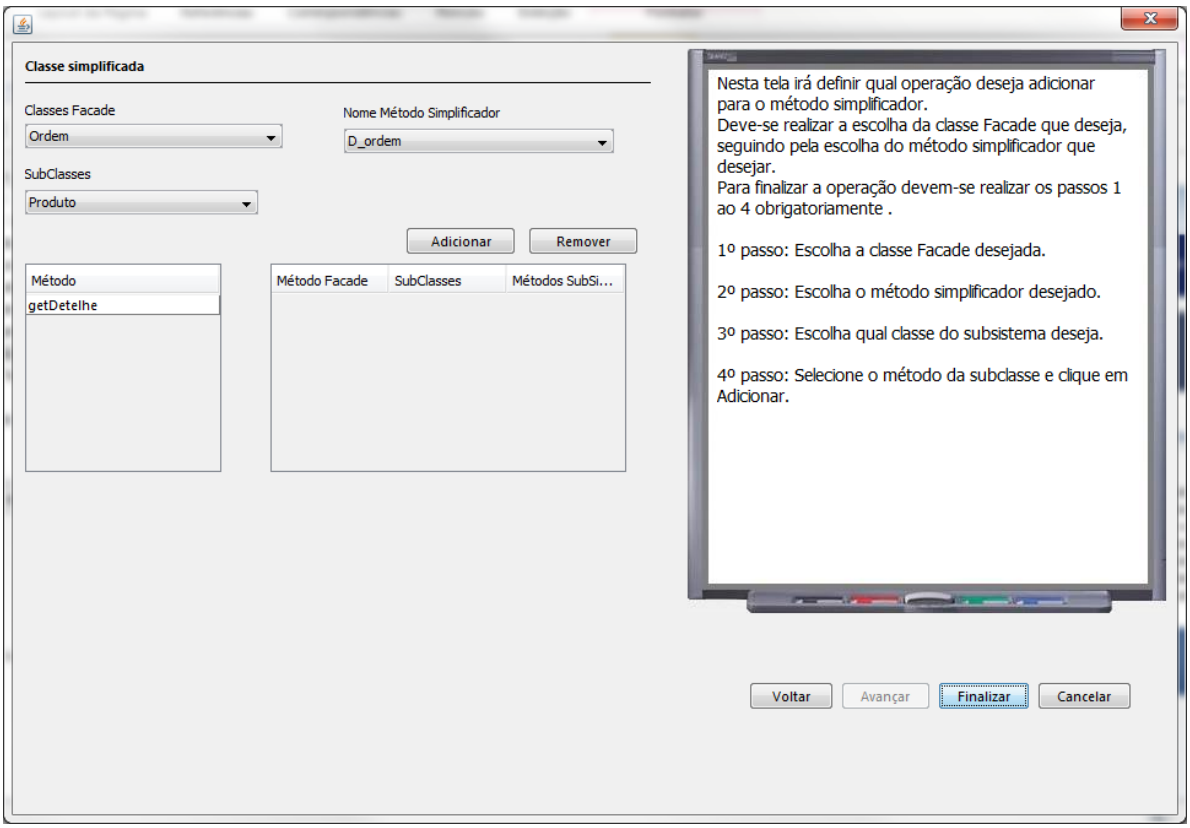


**Figura 17.** Tela do plugin para o padrão Facade: definições do método simplificador.

A partir do exemplo contido na Figura 13, o cliente fez um pedido sem ter conhecimento de como classes internas estão funcionando. Uma vez que a ordem é colocada a camada de classe fachada chama os métodos dos subsistemas, como dados de produto, forma de pagamento e fatura. Depois de processar ele retorna o controle para a classe do cliente com a confirmação sobre a ordem que está sendo processado. O *plugin* irá gerar os códigos apresentados nas Figuras 18, 19, 20 e 21.

A Figura 18 apresenta o código da classe Ordem, sendo que esta é a classe Facade, onde seu método simplificador é denominado de D_ordem.

O código gerado para a classe Fatura é apresentado na Figura 19.

```
public class Ordem
{
  private Produto produto;
  private Pagamento pagamento;
  private Fatura fatura;
  /*
  E definida classe Facade onde na mesma contém os métodos simplificadores    declarados
durante o processo de criação na ferramenta.
  Em cada método simplificador e delegado responsabilidade ao componente correspondente
no subsistema.
  O construtor receberá referência para cada componente do subsistema.
*/
  public Ordem(Produto produto,Pagamento pagamento,Fatura fatura)
  {
    this.produto = produto;
    this.pagamento = pagamento;
    this.fatura = fatura;
  }
  public void D_ordem()
  {
    produto.getDetalhe();
    pagamento.online();
    fatura.imprimir();
  }
}
```

**Figura 18.** Código-fonte gerado para a Classe Ordem

```
public class Fatura
{
    /*Esta classe pertence ao subsistema, onde na mesma contém os seus respectivos
métodos, parâmetros e tipos de retorno do mesmo declarados durante o processo de
criação.*/
   public  Fatura(){}
   public void imprimir()
   {
       throw new UnsupportedOperationException("Not supported yet.");
   }
}
```

**Figura 19.** Código-fonte gerado para a Classe Fatura

A Figura 20 apresenta o código gerado para a classe Pagamento. O código gerado para a classe Fatura é apresentado na Figura 21.

```
public class Pagamento
{
      /*Esta classe pertence ao subsistema, onde na mesma contém os seus respectivos métodos, parâmetros e tipos
de retorno do mesmo declarados durante o processo de criação.*/
    public  Pagamento()
    {}
    public void online()
    {
       throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

**Figura 20.** Código-fonte gerado para a Classe Pagamento

```
public class Produto
{
    /*Esta classe pertence ao subsistema, onde na mesma contém os seus respectivos métodos,
    parâmetros e tipos de retorno do mesmo declarados durante o processo de criação.*/
    public  Produto()
    {}
    public void getDetalhe()
    {
       throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

**Figura 21.** Código-fonte gerado para a Classe Pagamento.

## 5 ANÁLISE DOS RESULTADOS

Foi realizado uma análise preliminar da ferramenta e sua utilização durante as aulas de conteúdo de padrões de projeto no curso de Sistemas para Internet. O objetivo da analise foi avaliar a eficiência quanto a melhoria na aprendizagem.

Após utilizar a ferramenta os alunos responderam um questionário referente ao padrão de projeto aprendido e outras questões para análise da ferramenta. Foi possível observar como resultados preliminares obtidos: melhor entedimento quanto aos conceitos de padrões de projetos, e melhor entendimento do padrão e sua aplicabilidade.

As Tabelas 2, 3 e 4, apresentam as respostas obtidas pelos alunos em uma análise da ferramenta para o padrão Strategy.

A Tabela 2 apresenta um alto percentual de respostas corretas, aproximadamente oitenta e três por cento (80%), para a pergunta que objetiva validar o entendimento e propósito do padrão Strategy.

**Tabela 2.** Resultados obtidos para a pergunta de definição do padrão strategy.

| ***Qual a fefinição e propósito do padrão strategy?*** | | |
|---|---|---|
| *Alternativa* | *Total de respotas* | *Percentual* |
| Define uma família de algoritmos, encapsula cada um deles, deixando cada um variar independentemente. | 4 | 80% |
| Define a dependência um-para-muitos entre objetos para que quanto um objeto mude de estado todos os seus dependentes sejam avisados e atualizados automaticamente. | 1 | 20% |
| Anexa responsabilidadedades adicionais a um objeto dinamicamente. Os decoradores fornecem uma alternativa flexível de subclasse para estender a funcionalidade | 0 | 0% |

A Tabela 3 apresenta as respostas obtidas sobre o funcionamento do padrão Strategy onde é possível perceber que dentro de cinco resposta foi obtido um total de duas repostas corretas.

**Tabela 3.** Resultados obtidos para a pergunta de funcionamento do padrão strategy.

| ***Explique com suas palavras o funcionamento do Padrão Strategy.*** | |
|---|---|
| *Resposta* | *Correta* |
| Ele encapsula o algoritmo. | Não |
| Funciona no encapsulamento de algoritmos. | Não |
| Define novas operações da classe sem alterar os métodos. | Sim |
| O padrão Strategy faz com que o usuário possa criar métodos que sejam implementados e podendo ser abstrato ou não. Garantindo que seja implementado uma classe para implementá-los individualmente. É possível criar, também, uma classe de interface para comunicar com as classes que possuem os métodos. | Sim |
| É uma interface, que controla outras interfaces. | Não |

A Tabela 4 apresenta as respostas obtidas sobre os benefícios do encapsulamento sendo possível observar que 60 por cento de repostas estão corretas.

## 6 CONSIDERAÇÕES FINAIS

Foi observado no mercado que não se encontra ferramenta com o mesmo propósito de ensino tal qual a ferramenta desenvolvida oferece.

Durante a realização deste trabalho foram encontradas algumas dificuldades quanto a compreensão e abstração dos padrões de projeto desenvolvidos. A proposta de uma solução de interface com o usuário considerando usabilidade foi outra dificuldade.

**Tabela 4**. Resultados obtidos para a pergunta de benefícios do encapsulamento.

| ***Qual o benefício do encapsulamento?*** | |
|---|---|
| *Resposta* | *Correta* |
| O acesso controlado dos métodos disponíveis para cada classe. | Sim |
| Reutilizar métodos e proteger. | Não |
| Encapsulamento é uma forma de juntar tudo, como uma cápsula, facilitando a "conversação do código." | Sim |
| Tornar os métodos intercambiáveis. | Não |
| Esconder todos os detalhes do código. | Sim |

Com o estudo observou-se que a utilização da ferramenta durante os procedimentos de ensino e aprendizagem de padrões facilitou o entendimento dos conceitos de orientação a objetos utilizados pelo padrão, entendimento dos conceitos de padrões de projeto, entendimento do padrão de projeto Strategy e sua aplicabilidade.

Como trabalhos futuros fica a proposta da implementação do restante dos padrões de projeto da linguagem de padrões GOF, proposta por Gamma et al. (1995).

É necessário, ainda, a realização de testes para avaliar a usabilidade da ferramenta e propor melhorias em sua interface com objetivo de tornar o processo mais interativo ao usuário.

Torna-se necessário, ainda, para complementar os resultados perante aos objetivos da ferramenta replicar a avaliação em um grupo maior de discentes.

## 7 REFERÊNCIAS

DELLA, L.; CLARK, D. Teaching object-oriented development with emphasis on pattern application. In: THE AUSTRALASIAN CONFERENCE ON COMPUTING EDUCATION. **Proceedings…**

Melbourne, Australia, 2000, p. 56-63. http://dx.doi.org/10.1145/359369.359378

FREEMAN, E.; FREEMAN, E. **Use a cabeça!** Padrões de projetos. 2. ed. Rio de Janeiro: Alta BooKs, 2009.

GAMMA, E. et al. **Design patterns:** elements of reusable object oriented software. Addison-Wesley, 1995.

GUERRA, E. **Design patterns com Java:** projeto orientado a objetos guiado por padrões. São Paulo: Casa do Código, 2013.